



Hack Yourself First

Web Application Security



Agenda

1. OWASP
2. Top 10 Web Application Security Risks
3. Injection (SQL, NoSQL, OS, LDAP)
4. Broken Authentication
5. Sensitive Data exposure
6. Broken Access control
7. Security Misconfiguration
8. Cross-Site Scripting (XSS)
9. Insecure Deserialization
10. Insecure Direct Object References

Demo

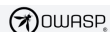


<https://github.com/aglumova/web-security-workshop>

OWASP

Open Web Application Security Project

<https://owasp.org>



PROJECTS CHAPTERS EVENTS ABOUT

Search OWASP.org

Donate

Join



Who is the OWASP® Foundation?

The Open Web Application Security Project® (OWASP) is a nonprofit foundation that works to improve the security of software. Through community-led open-source software projects, hundreds of local chapters worldwide, tens of thousands of members, and leading educational and training conferences, the OWASP Foundation is the source for developers and technologists to secure the web.

- Tools and Resources
- Community and Networking
- Education & Training

For nearly two decades corporations, foundations, developers, and volunteers have supported the OWASP Foundation and its work. [Donate](#), [Join](#), or become a [Corporate Member](#) today.

Project Spotlight: Security Knowledge Framework



The OWASP Security Knowledge Framework (SKF) is a fully open-source Python-Flask web application that uses

Featured Chapter: Sydney

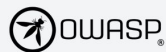


OWASP Sydney Chapter

Leadership are on fire! Ric Campo, Jack Guildford, and

Top 10 Web Application Security Risks

<https://owasp.org/www-project-top-ten/>



PROJECTS CHAPTERS EVENTS ABOUT

OWASP Top Ten

[Main](#)

[Translation Efforts](#)

[Sponsors](#)

[Data 2020](#)

The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

Globally recognized by developers as the first step towards
more secure coding.

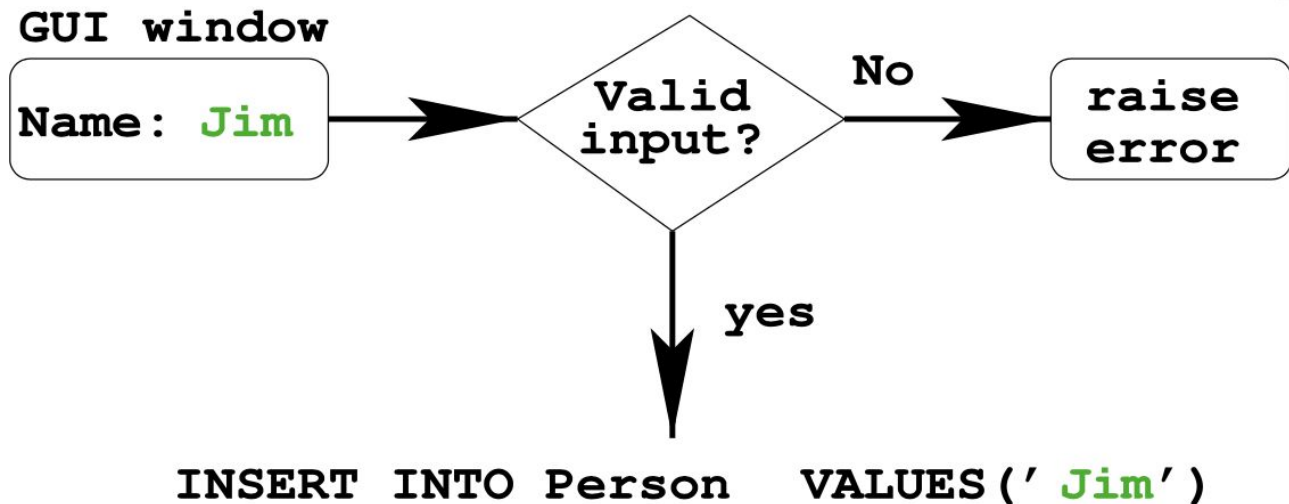
Companies should adopt this document and start the process of ensuring that their web applications minimize these risks. Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organization into one that produces more secure code.

Injection



Injection (SQL, NoSQL, OS, LDAP)

Classic failure to filter untrusted input.



Prevention:

- White list filtering
- Trust your frameworks

SQL Injection

Java Example



Application

JDBC Statement

```
INSERT INTO ...  
VALUES ('Jim',  
       'jim@q.com')
```

RDBMS

Interpreter

Low-level
calls, AST

Application

PreparedStatement

```
INSERT INTO ...  
VALUES (?, ?)
```

RDBMS

Interpreter

AST

Jim

jim@f.com

p.setString(1, "Jim")

p.setString(2, "jim@f.com")

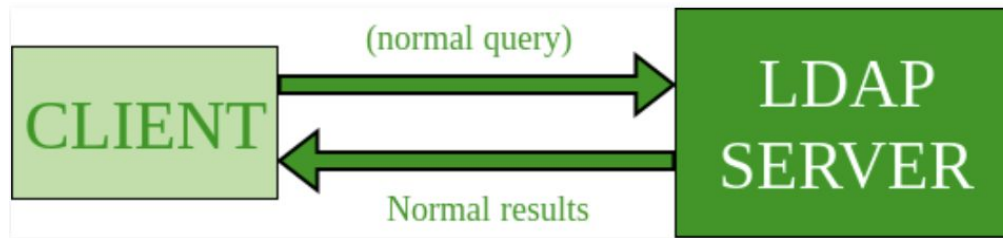
Finally:

p.executeUpdate()

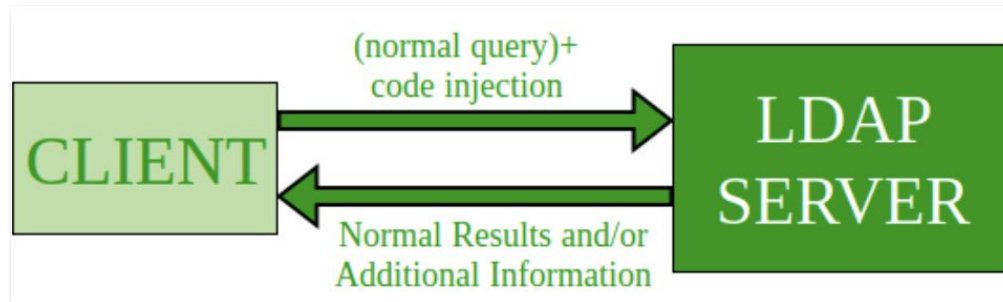
2 Placeholders

LDAP Injection

Normal Operation:



Operation with **Code Injection**:



Broken Authentication



Broken Authentication

1. The URL might contain the session id and leak it in the referer header to someone else.
2. The passwords might not be encrypted either in storage or transit.
3. The session ids might be predictable, thus gaining access is trivial.
4. Session hijacking might be possible, timeouts not implemented right or using HTTP (no SSL security).
5. Etc.

Prevention:

- **Straightforward:** Trust your frameworks
- **Roll your own code:** Be extremely paranoid and educate yourself on what the pitfalls are

How to Store Password?

Worst: Store password as plain text

Bad: Obsolete Hash Function (md5, sha1...)

The LinkedIn site used to store part of its passwords with sha1, and after the hash leaks in 2012, it took only three days to recover 90% of the passwords.

Login	Password (md5)
admin	ab4f63f9ac65152575886860dde480a1
toto	21b72c0b7adc5c7b4a50ffcb90d92dd6
billy	47ad898a379c3dad10b4812eba843601
tata	21b72c0b7adc5c7b4a50ffcb90d92dd6
titi	5b9a8069d33fe9812dc8310ebff0a315



A screenshot of a Google search interface. The search bar contains the MD5 hash 'ab4f63f9ac65152575886860dde480a1'. Below the search bar, the results show several links to websites that can reverse MD5 hashes. The first result is from 'md5.gromweb.com' with the title 'MD5 reverse for ab4f63f9ac65152575886860dde480a1'. The second result is from 'hashtoolkit.com' with the title 'ab4f63f9ac65152575886860dde480a1 - Hash Toolkit'. The third result is from 'md5calc.com' with the title 'MD5 hash for "azerty" is ... - Md5Calc.com'. The fourth result is from 'md5hashing.net' with the title 'Hash Md5: ab4f63f9ac65152575886860dde480a1'. The search results also indicate that the hash was successfully reversed into the string 'azerty'.

How to Store Password?

Bad: Inappropriate hash function (sha256, sha512, or sha3)

They are not suitable for storing passwords, because they are fast to calculate, as the following benchmark proves

Normal: Improving SHA512: Static and Dynamic Salt

User	Salt	Hash sha512 with salt
admin	BGdd6d6^ZgvkMhKf@W3RqT	7509d123bce1aa92331861cf8fd738a58205045123f0e25f
toto	HZBD^@gL*wvoExo6yJ7hVB	6b28830776de6ad7ef1dd8c221e0d53fec4532c623075d02
billy	wvVndjwcZJy!dwT4fBD@U^	2847b2605f6a1cd88399e6c9784c0e583799be9485cb128f
tata	QeNWm9NXqJ8m@m2^F7Kh9*	165bc06b69fa2bfcd893bfde86358394406c87c7f7abba89
titi	iQUengw9M6Gw*&v6RG%MZ#	f8eded6c815c7522ab6197aa319d3ff4cddc2c7eeffa0f91

NOTE: In some resources Static Salt named - Pepper.

How to Store Password

Good: Increasing the Number of Iterations



As long as iteration is greater than 0

```
hash = sha512(hash)
```

```
Decrement iteration
```

Best:

- Merging the 3 Methods



- BCrypt

```
Function calculation_hash(password,salt,pepper,iteration)
```

Inputs

password is the user's password in plain text

salt is the unique salt per user and is randomly generated

pepper is the common pepper for all users and is randomly generated.

iteration is the number of iterations

Output:

The password hash

```
Hash = sha512(salt+password+pepper)
```

As long as iteration is greater than 0

```
hash = sha512(hash)
```

```
Decrement iteration
```

```
return hash
```

BCrypt

The hash computed by bcrypt has a predefined form:

\$2y\$11\$SXAXZyioy60hbnymeoj9.ulscXwUFMhbvLaTxAt729tGusw.5AG4C

1. **Algorithm:** This one can take several versions depending on the version of bcrypt (\$2\$, \$2a\$, \$2x\$, \$2y\$ and \$2b\$)
2. **The cost:** The number of iterations in power of 2.
3. **Salt:** Instead of storing the salt in a dedicated column, it is directly stored in the final hash.

Session hijacking



Session hijacking

Three common protection strategies against session fixation

- **Only use HTTPS**

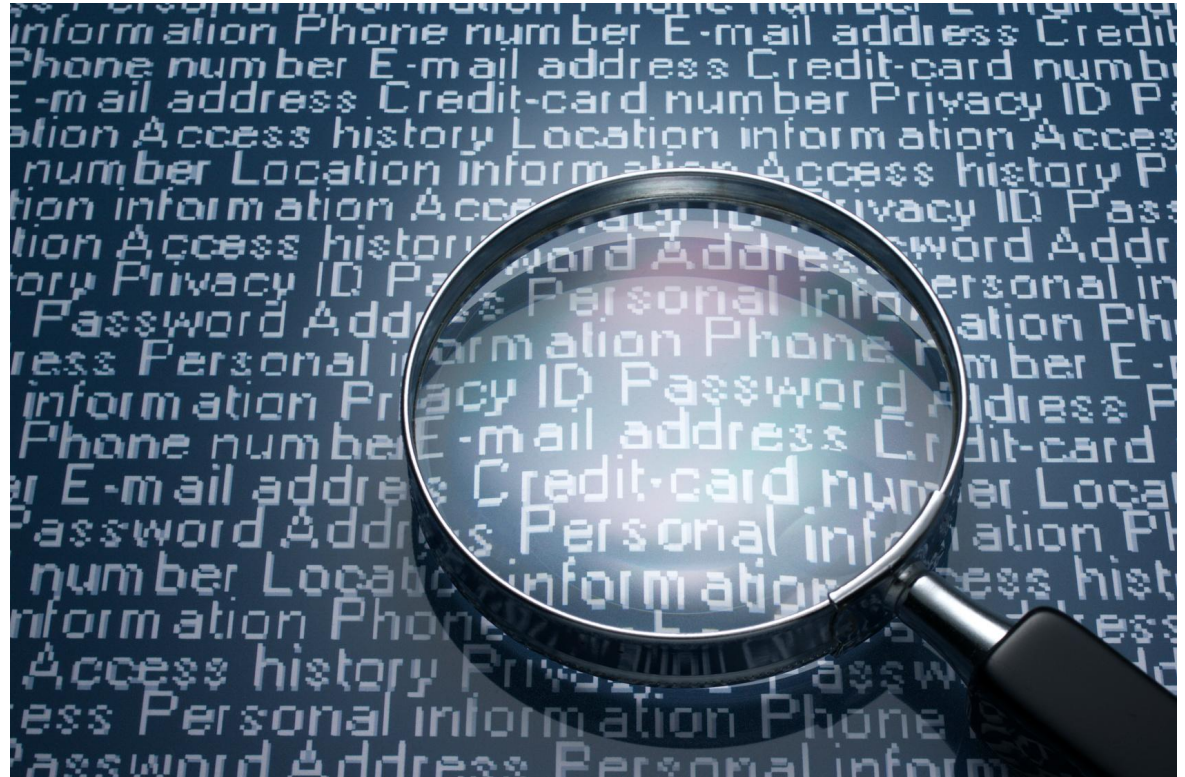
Downside:

- Increases processing overhead
- Increases network traffic
- Makes caches much less effective.

- **Once a user logs in, enforce HTTPS for future traffic.**

- **Secure HTTPS traffic with a secondary cookie**

Sensitive Data Exposure



Sensitive Data Exposure

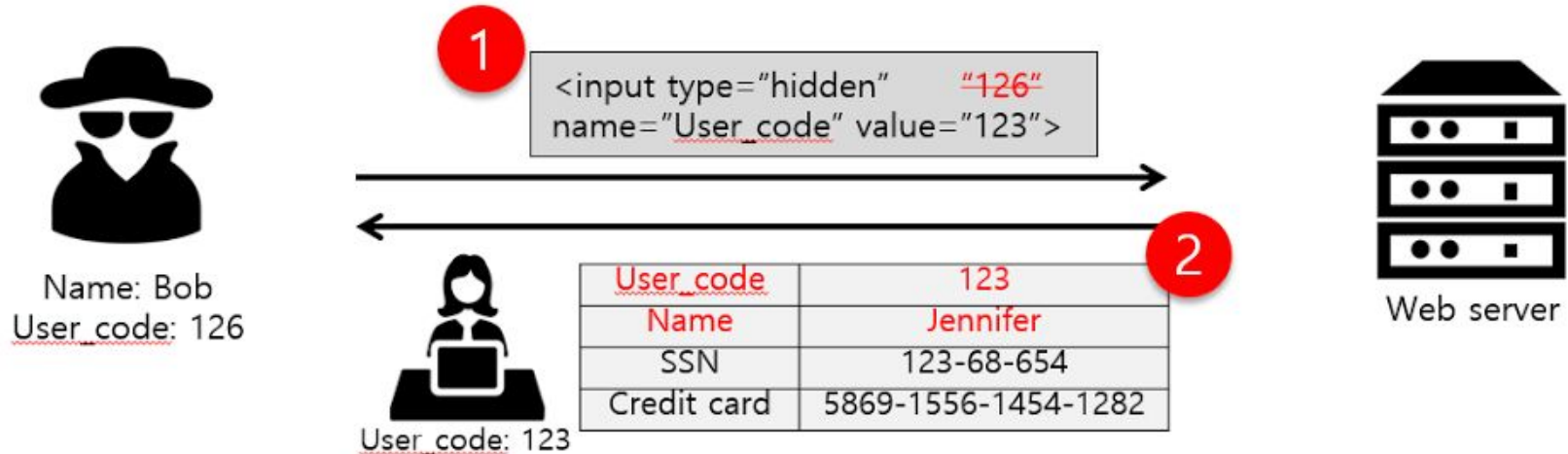
Sensitive data should be encrypted at all times, including in transit and at rest. No exceptions.

- session IDs and sensitive data **should not** be traveling in the URLs
- sensitive cookies **should have** the secure flag on

Prevention:

- *In transit:* Use [HTTPS](#) with a proper certificate and [PFS \(Perfect Forward Secrecy\)](#). Do not accept anything over non-HTTPS connections. Have the secure flag on cookies.
- *In storage:* **First** and foremost, you need to lower your exposure. Do not store credit card information *ever*, as you probably don't want to have to deal with being [PCI compliant](#). Sign up with a payment processor such as [Stripe](#) or [Braintree](#). **Second**, if you have sensitive data that you actually do need, store it encrypted and make sure all passwords are hashed. For hashing, use of [bcrypt](#) is recommended. If you don't use bcrypt, educate yourself on [salting](#) and [rainbow tables](#).

Broken Access Control




Broken Access Control

- The application uses unverified data



```
http://example.com/app/accountInfo?acct=notmyacct
```

- An attacker simply force browses to target URLs



```
http://example.com/app/getappInfo  
http://example.com/app/admin_getappInfo
```

Prevention:

- Deny access to functionality by default.
- Use Access control lists and role-based authentication mechanisms.
- **!**Do not just hide functions.

Security Misconfiguration



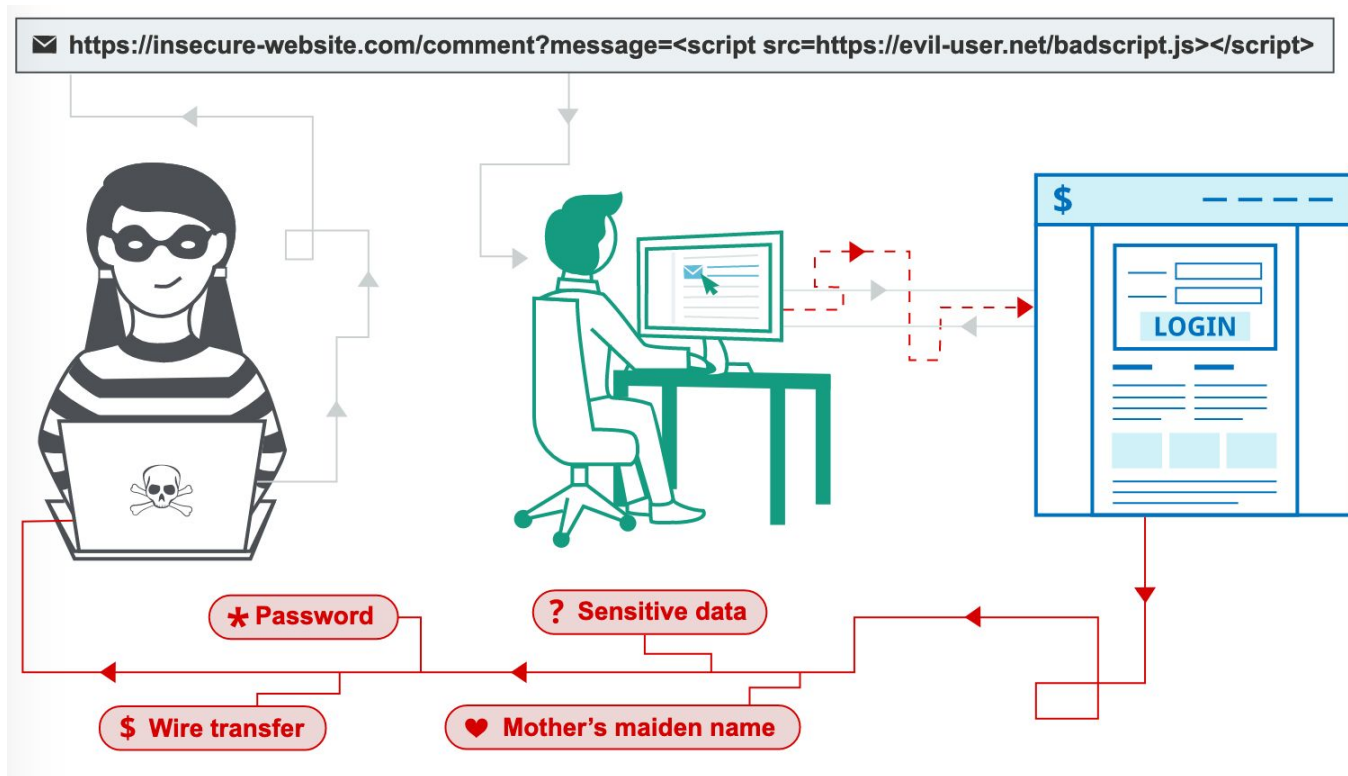
Security Misconfiguration

1. Running the application with debug enabled in production.
2. Having directory listing enabled on the server, which leaks valuable information.
3. Running outdated software.
4. Having unnecessary services running on the machine.
5. Not changing default keys and passwords. (Happens way more frequently than you'd believe!)
6. Revealing error handling information to the attackers, such as stack traces.

Prevention:

- Have a good “build and deploy” process, which can **run tests on deploy**.
- The poor man's security misconfiguration solution is post-commit hooks, to **prevent the code from going out with default passwords** and/or development stuff built in

Cross-Site Scripting (XSS)



Cross-Site Scripting (XSS)

- **Reflected XSS**, where the malicious script comes from the current HTTP request.

```
https://insecure-website.com/status?message=<script>/*Bad+stuff+here...+*/</script>  
<p>Status: <script>/* Bad stuff here... */</script></p>
```

- **Stored XSS**, where the malicious script comes from the website's database.
- **DOM-based XSS**, where the vulnerability exists in client-side code rather than server-side code.

```
var search = document.getElementById('search').value;  
var results = document.getElementById('results');  
results.innerHTML = 'You searched for: ' + search;
```



```
You searched for: <img src=1 onerror='/* Bad stuff here... */'>
```

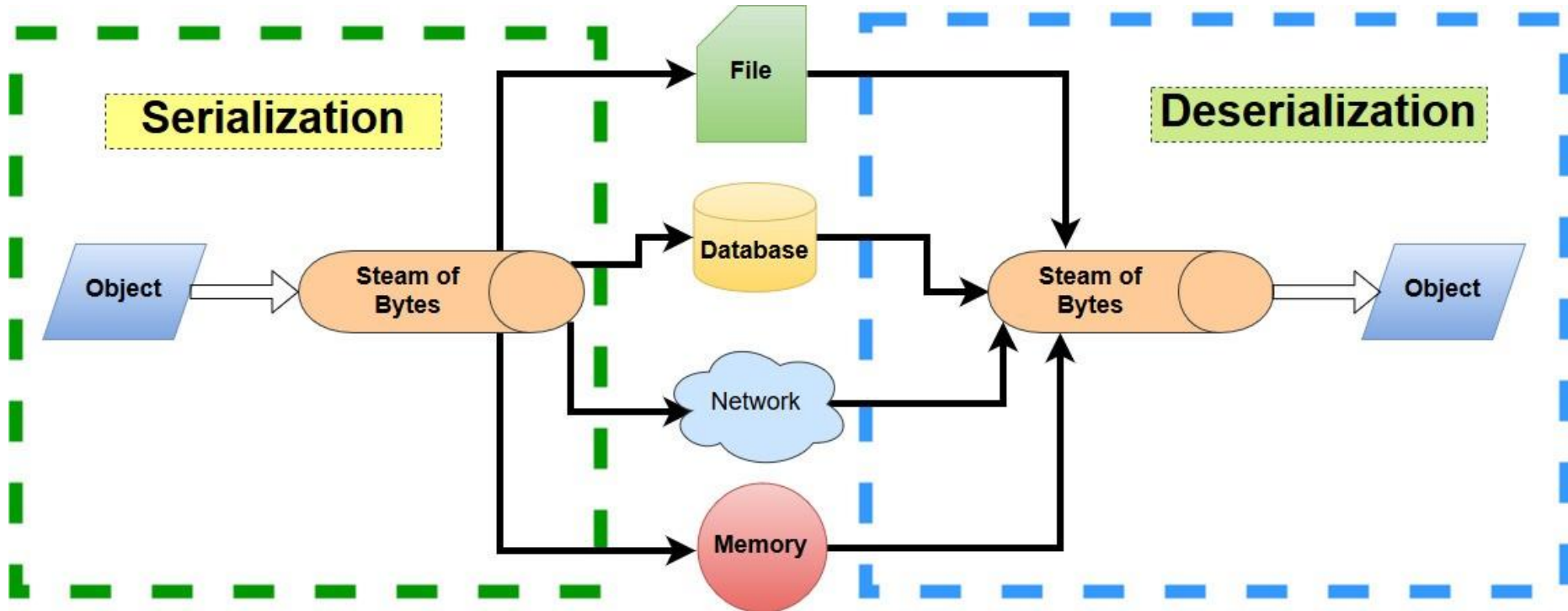
Cross-Site Scripting (XSS)

Prevention:

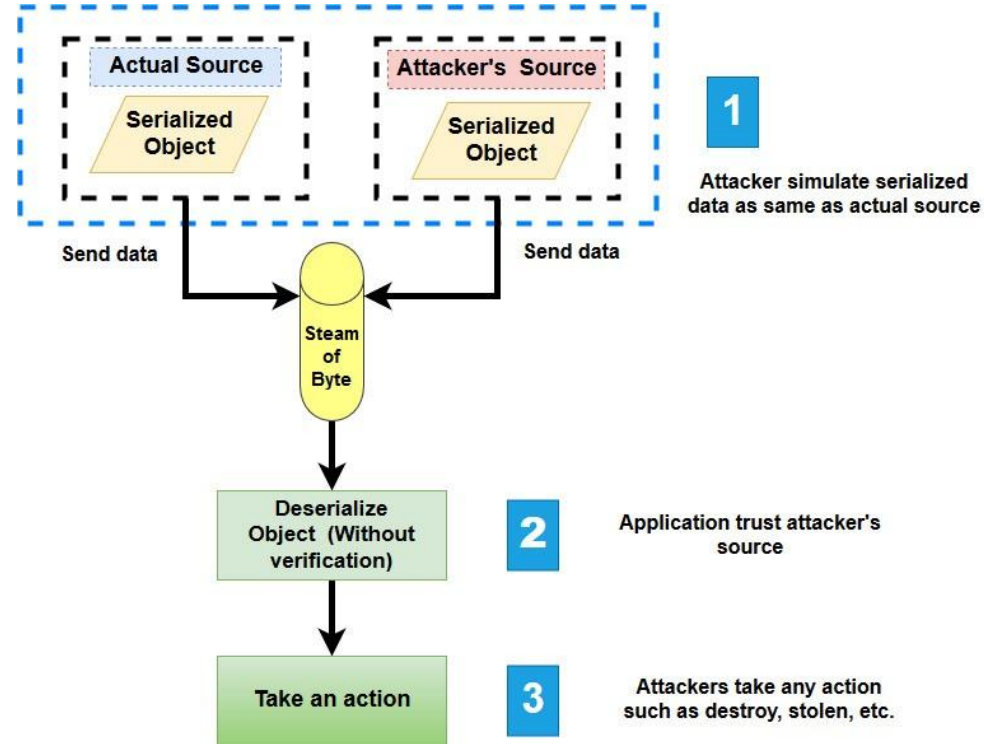
- Filter input on arrival.
- Encode data on output.
- Use appropriate response headers

Content-Type and **X-Content-Type-Options** headers to ensure that browsers interpret the responses in the way you intend.

Insecure Deserialization



Insecure Deserialization



The vulnerability of applications always occurs from below

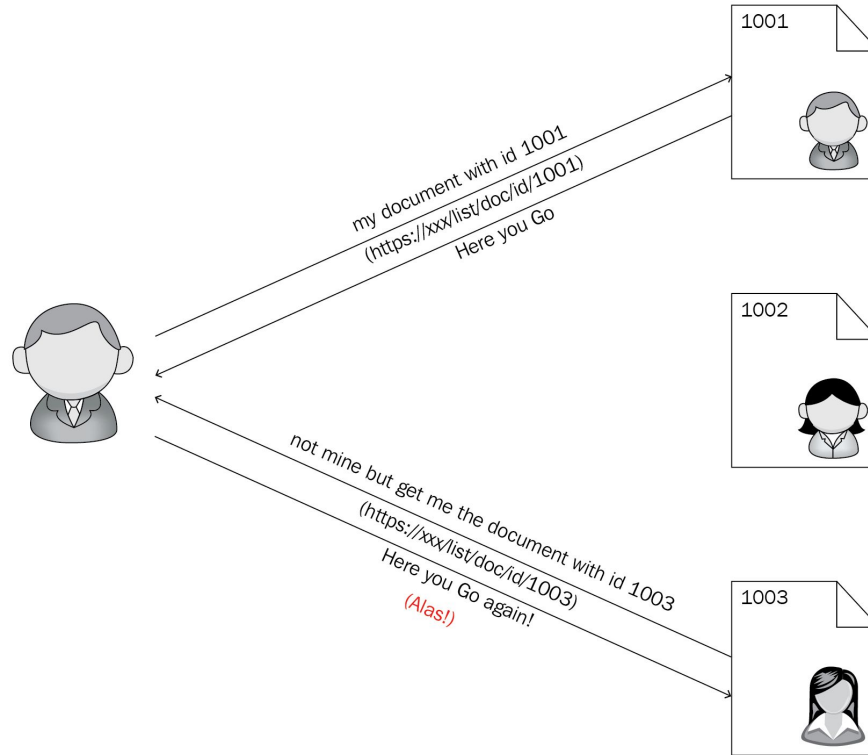
- read data from untrusted sources.
- read data without verification such as digital signature, unsafe classes

Insecure Deserialization

Prevention:

- Monitoring the deserialization process.
- Encrypting serialization processes.
- Not accepting serialized objects from unknown or untrusted sources.
- Running the deserialization code with limited access permissions.
- Using a firewall which can help detect insecure deserialization

Insecure Direct Object Reference



Insecure Direct Object Reference

Prevention:

- Do not use predicted ID's outside the internal communication

From `userId=1<Number>` → To `userId=e45571e1-4cd8-4ebe-b051-f4fa17614807<GUID>`

- Strict access control checks to all resources 